5 Automating deployment: CloudFormation, Elastic Beanstalk, and OpsWorks

This chapter covers

- Creating VMs and running scripts on startup with AWS CloudFormation
- Deploying common web apps with AWS Elastic Beanstalk
- Deploying multilayer apps with AWS OpsWorks
- Comparing the different deployment services on AWS

Whether you want to use software from in-house development, open source projects, or commercial vendors, you need to install, update, and configure the application and its dependencies. This process is called *deployment*. In this chapter, you'll learn about three tools for deploying applications to virtual machines on AWS:

1. Deploying a VPN solution with the help of AWS CloudFormation and a script that starts at the end of the boot process.

2. Deploying a collaborative text editor with AWS Elastic Beanstalk. The text editor *Etherpad* is a simple web application and a perfect fit for AWS Elastic Beanstalk, because it supports Node.js by default.
3. Deploying an IRC web client and IRC server with AWS OpsWorks. The setup consists of two parts: the IRC web client and the IRC server itself. Our example consists of multiple layers and is perfect for AWS OpsWorks.

We've chosen examples that don't need a storage solution for this chapter, but all three deployment solutions would support delivering an application together with a storage solution. You'll find examples using storage in part 3 of the book.

Examples are 100% covered by the Free Tier

The examples in this chapter are completely covered by the Free Tier. As long as you don't run the examples longer than a few days, you won't pay anything. Keep in mind that this only applies if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

What steps are required to deploy a typical web application like WordPress—a widely used blogging platform—to a virtual machine?

1. Install an Apache HTTP server, a MySQL database, a PHP runtime environment, a MySQL library for PHP, and an SMTP mail server.
2. Download the WordPress application, and unpack the archive on your server.
3. Configure the Apache web server to serve the PHP application.
4. Configure the PHP runtime environment to tweak performance and increase security.
5. Edit the wp-config.php file to configure the WordPress application.
6. Edit the configuration of the SMTP server, and make sure mail can only be sent from the virtual machine, to avoid misuse from spammers.
7. Start the MySQL, SMTP, and HTTP services.

Steps 1–2 handle installing and updating the executables. These executables are configured in steps 3–6. Step 7 starts the services.

System administrators working with a traditional infrastructure often perform these steps manually by following how-to guides. Deploying applications manually is no longer recommended in a flexible cloud environment. Instead your goal will be to automate these steps with the help of the tools you'll discover next.

5.1 Deploying applications in a flexible cloud environment

If you want to take advantage of cloud features like scaling the number of machines depending on the current load or building a highly available infrastructure, you'll need to start new virtual machines several times a day. On top of that, the number of

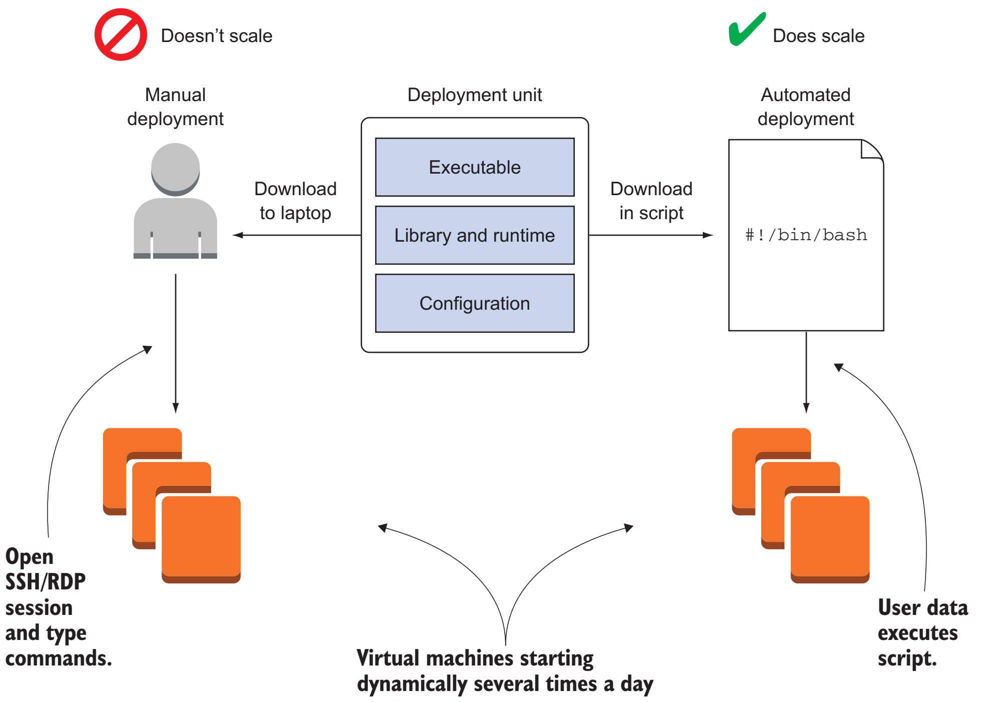


Figure 5.1 Deployment must be automated in a flexible and scalable cloud environment.

VMs you'll have to keep updated will grow. The steps required to deploy an application don't change, but as figure 5.1 shows, you need to perform them on multiple VMs. Deploying software manually to a growing number of VMs becomes impossible over time and has a high risk of human failure. This is why we recommend that you automate the deployment of applications.

The investment in an automated deployment process will pay off in the future by increasing efficiency and decreasing human error. In the next section, you will learn about options for automation that you will examine in more detail throughout the rest of the chapter.

5.2 Comparing deployment tools

You will learn about three ways to deploy an application in this chapter:

1. Creating a virtual machine and running a deployment script on startup with AWS CloudFormation.
2. Using AWS Elastic Beanstalk to deploy a common web application.
3. Using AWS OpsWorks to deploy a multilayer application.

In this section, we'll discuss the differences between these solutions. After that, you will take a dive deep into each.

5.2.1 Classifying the deployment tools

Figure 5.2 depicts the three AWS deployment options. The effort required to deploy an application using AWS Elastic Beanstalk is low. To benefit from this, your application has to fit into the conventions of AWS Elastic Beanstalk. For example, the application must run in one of the standardized runtime environments. If you're using OpsWorks Stacks, you'll have more freedom to adapt the service to your application's needs. For example, you can deploy different layers that depend on each other, or you can use a custom layer to deploy any application with the help of a *Chef recipe*; this takes extra effort but gives you additional freedom. On the other end of the spectrum you'll find CloudFormation and deploying applications with the help of a script that runs at the end of the boot process. You can deploy any application with the help of CloudFormation. The disadvantage is that you have to do more work, because you don't use standard tooling.

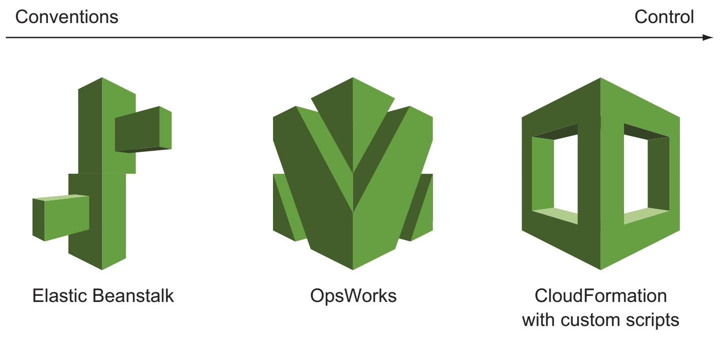


Figure 5.2 Comparing different ways to deploy applications on AWS

5.2.2 Comparing the deployment services

The classification in the previous section can help you decide the best fit to deploy an application. The comparison in table 5.1 highlights other important considerations.

Table 5.1 Differences between using CloudFormation with a script on virtual machine startup, Elastic Beanstalk, and OpsWorks Stacks

	CloudFormation with a script on VM startup	Elastic Beanstalk	OpsWorks
Configuration management tool	All available tools	Proprietary	Chef
Supported platforms	Any	▪ PHP ▪ Node.js ▪ .NET on Windows Server with IIS ▪ Java (SE or Tomcat) ▪ Python ▪ Ruby ▪ Go ▪ Docker	▪ PHP ▪ Node.js ▪ Java (Tomcat) ▪ Ruby on Rails ▪ Custom / Any

Table 5.1 Differences between using CloudFormation with a script on virtual machine startup, Elastic Beanstalk, and OpsWorks Stacks *(continued)*

	CloudFormation with a script on VM startup	Elastic Beanstalk	OpsWorks
Supported deployment artifacts	Anything	Zip archive on Amazon S3	Git, SVN, archive (such as Zip)
Common scenario	Medium- to enterprise-sized companies	Small companies	Companies with prior experience using Chef
Update without downtime	Not by default, but possible	Yes	Yes
Vendor lock-in effect	Medium	High	Medium

Many other options are available for deploying applications on AWS, from open source software to third-party services. Our advice is to use one of the AWS deployment services because they're well integrated into many other AWS services. We recommend that you use AWS CloudFormation with user data to deploy applications, because it's a flexible approach.

An automated deployment process will help you to iterate and innovate more quickly. You'll deploy new versions of your applications more often. To avoid service interruptions, you need to think about testing changes to software and infrastructure in an automated way, and being able to roll back to a previous version quickly if necessary.

In the next section you will use a Bash script and CloudFormation to deploy an application.

5.3 Creating a virtual machine and run a deployment script on startup with AWS CloudFormation

A simple but powerful and flexible way to automate application deployment is to launch a virtual machine and then launch a script at startup. To go from a plain OS to a fully installed and configured VM, follow these steps:

1. Start a plain virtual machine containing just an OS.
2. Execute a script the end of the boot process.
3. Install and configure your applications with the help of the script.

First you need to choose an AMI from which to start your virtual machine. An AMI bundles the OS for your VM with preinstalled software. When you start your VM from an AMI containing a plain OS without any additional software installed, you need to provision the VM at the end of the boot process. Otherwise all your virtual machines will look the same and will run a plain OS, which is not very helpful. You want to install custom applications, not a plain OS. Translating the necessary steps to install and configure your application into a script allows you to automate this task. But how do you execute this script automatically after booting your virtual machine?

5.3.1 Using user data to run a script on startup

You can inject a small amount of data called *user data*—no more than 16 KB— into every VM to customize them besides what comes in the AMI. You specify this user data during the creation of a new VM and can query it later from the machine itself. A typical way of using this feature is built into most AMIs, such as the Amazon Linux Image and the Ubuntu AMI. Whenever you boot a VM based on these AMIs, user data is executed as a shell script at the end of the boot process. The script is executed as the root user.

The user data is always accessible from the VM with a HTTP `GET` request to http://169.254.169.254/latest/user-data. The user data behind this URL is only accessible from the VM itself. As you'll see in the following example, you can deploy applications of any kind with the help of user data executed as a script.

5.3.2 Deploying OpenSwan: a VPN server to a virtual machine

If you're working over public Wi-Fi, for example using a laptop at a coffee house, you may want to tunnel your traffic through a VPN because unencrypted communication (such as HTTP instead of HTTPS) can be intercepted by an attacker. You'll learn how to deploy a VPN server to a virtual machine with the help of user data and a shell script next. The VPN solution, called *OpenSwan*, offers an IPSec-based tunnel that's easy to use with Windows, macOS, and Linux. Figure 5.3 shows the example setup.

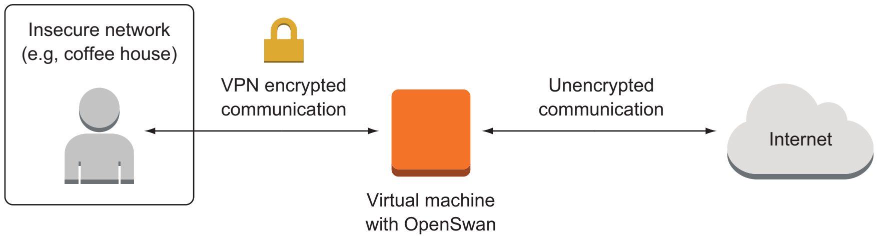


Figure 5.3 Using OpenSwan on a virtual machine to tunnel traffic from a personal computer

Open your terminal, and execute the commands shown in listing 5.1 step by step to start a virtual machine and deploy a VPN server on it. We've prepared a CloudFormation template that starts the virtual machine and its dependencies.

Shortcut for Linux and macOS

You can avoid typing these commands manually at your terminal by using the following command to download a Bash script and execute it directly on your local machine. The Bash script contains the same steps as shown in listing 5.1.

```
$ curl -s https://raw.githubusercontent.com/AWSinAction/\
➥ code2/master/chapter05/\
➥ vpn-create-cloudformation-stack.sh | bash -ex
```

Listing 5.1 Deploying a VPN server to a virtual machine: CloudFormation and a shell script

Gets the default subnet

Gets the default VPC

```
$ VpcId="$(aws ec2 describe-vpcs --query "Vpcs[0].VpcId" --output text)"

$ SubnetId="$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=$VpcId" \
➥ --query "Subnets[0].SubnetId" --output text)"

$ SharedSecret="$(openssl rand -base64 30)"

$ Password="$(openssl rand -base64 30)"

$ aws cloudformation create-stack  --stack-name vpn --template-url \
➥ https://s3.amazonaws.com/awsinaction-code2/chapter05/\
➥ vpn-cloudformation.yaml \
➥ --parameters ParameterKey=KeyName,ParameterValue=mykey \
➥ "ParameterKey=VPC,ParameterValue=$VpcId" \
➥ "ParameterKey=Subnet,ParameterValue=$SubnetId" \
➥ "ParameterKey=IPSecSharedSecret,ParameterValue=$SharedSecret" \
➥ ParameterKey=VPNUser,ParameterValue=vpn \
➥ "ParameterKey=VPNPassword,ParameterValue=$Password"

aws cloudformation wait stack-create-complete --stack-name vpn

$ aws cloudformation describe-stacks --stack-name vpn \
➥ --query "Stacks[0].Outputs"
```

Creates a random shared secret. (If openssl is not working, create your own secret.)

Creates a CloudFormation stack

Creates a random password (if openssl is not working, create your own random sequence).

Wait until stack is CREATE_COMPLETE.

Get stack outputs.

The output of the last command should print out the public IP address of the VPN server, a shared secret, the VPN username, and the VPN password. You can use this information to establish a VPN connection from your computer, if you like:

```
[{
  "Description": "The username for the vpn connection",
  "OutputKey": "VPNUser",
  "OutputValue": "vpn"
}, {
  "Description": "The shared key for the VPN connection (IPSec)",
  "OutputKey": "IPSecSharedSecret",
  "OutputValue": "EtAYOHXaLjcJ9nLCLEBfkZ+qV3H4Jy3MMc03Ehfy"
}, {
  "Description": "Public IP address of the virtual machine",
  "OutputKey": "ServerIP",
  "OutputValue": "34.202.233.247"
}, {
  "Description": "The password for the vpn connection",
  "OutputKey": "VPNPassword",
  "OutputValue": "MXBOtTlx3boJV+2r3tlOs6MCQisMhcj8oLVLilO2"
}]
```

Let's take a deeper look at the deployment process of the VPN server. We'll examine the following tasks, which you've already used:

- Starting a virtual machine with custom user data and configuring a firewall for the VM with AWS CloudFormation.
- Executing a shell script at the end of the boot process to install an application and its dependencies with the help of a package manager, as well as to edit configuration files

USING CLOUDFORMATION TO START A VIRTUAL MACHINE WITH USER DATA

You can use CloudFormation to start a virtual machine and configure a firewall. The template for the VPN server includes a shell script packed into user data, as shown in listing 5.2.

!Sub and !Base64

The CloudFormation template includes two new functions `!Sub` and `!Base64`. With `!Sub`, all references within `${}` are substituted with their real value. The real value will be the value returned by `!Ref`, unless the reference contains a dot, in which case it will be the value returned by `!GetAtt`:

```
!Sub 'Your VPC ID: ${VPC}' # becomes 'Your VPC ID: vpc-123456'
!Sub '${VPC}' # is the same as !Ref VPC
!Sub '${VPC.CidrBlock}' # is the same as !GetAtt 'VPC.CidrBlock'
!Sub '${!VPC}' # is the same as '${VPC}'
```

The function `!Base64` encodes the input with Base64. You'll need this function because the user data must be encoded in Base64:

```
!Base64 'value' # becomes 'dmFsdWU='
```

Listing 5.2 Parts of a CloudFormation template to start a virtual machine with user data

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'AWS in Action: chapter 5 (OpenSwan acting as VPN IPSec endpoint)'
Parameters:                                              <- Parameters to make it possible
  KeyName:                                                  to reuse the template
    Description: 'Key pair name for SSH access'
    Type: 'AWS::EC2::KeyPair::KeyName'
  VPC:
    Description: 'Just select the one and only default VPC.'
    Type: 'AWS::EC2::VPC::Id'
  Subnet:
    Description: 'Just select one of the available subnets.'
    Type: 'AWS::EC2::Subnet::Id'
  IPSecSharedSecret:
    Description: 'The shared secret key for IPSec.'
    Type: String
  VPNUser:
    Description: 'The VPN user.'
    Type: String
  VPNPassword:
    Description: 'The VPN password.'
    Type: String
Resources:
  EC2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: 'ami-6057e21a'
      InstanceType: 't2.micro'
      KeyName: !Ref KeyName
      NetworkInterfaces:
      - AssociatePublicIpAddress: true
        DeleteOnTermination: true
        DeviceIndex: 0
        GroupSet:
        - !Ref InstanceSecurityGroup
        SubnetId: !Ref Subnet
      UserData:
        'Fn::Base64': !Sub |
          #!/bin/bash -x
          export IPSEC_PSK="${IPSecSharedSecret}"
          export VPN_USER="${VPNUser}"
          export VPN_PASSWORD="${VPNPassword}"
          curl -s https://raw.githubusercontent.com/AWSinAction/code2/\
➥ master/chapter05/vpn-setup.sh | bash -ex
          /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} \
➥ --resource EC2Instance --region ${AWS::Region}
    CreationPolicy:
      ResourceSignal:
        Timeout: PT10M
  InstanceSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'Enable access to VPN server'
      VpcId: !Ref VPC
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: '0.0.0.0/0'
      - IpProtocol: udp
        FromPort: 500
        ToPort: 500
        CidrIp: '0.0.0.0/0'
      - IpProtocol: udp
        FromPort: 1701
        ToPort: 1701
        CidrIp: '0.0.0.0/0'
      - IpProtocol: udp
        FromPort: 4500
        ToPort: 4500
        CidrIp: '0.0.0.0/0'
Outputs:
  # [...]
```

Describes the virtual machine

Defines a shell script as user data for the virtual machine

Substitutes and encodes a multi-line string value

Exports parameters to environment variables to make them available in an external shell script called next

Fetches the shell script via HTTP and executes it

CloudFormation will wait up to 10 minutes to receive a signal via the cfn-signal tool that runs in user data.

Signals end of script back to Cloud-Formation

The user data contains a small script to fetch and execute the real script, vpn-setup.sh, which contains all the commands for installing the executables and configuring the services. Doing so frees you from inserting complicated scripts in the CloudFormation template.

INSTALLING AND CONFIGURING A VPN SERVER WITH A SCRIPT

The vpn-setup.sh script shown in the following listing installs packages with the help of the package manager `yum` and writes some configuration files. You don't have to understand the details of the VPN server configuration; you only need to know that this shell script is executed during the boot process to install and configure a VPN server.

Listing 5.3 Installing packages and writing configuration files on virtual machine startup

```
#!/bin/bash -ex

[...]
                                              Fetches the private IP address
                                                      of the virtual machine

PRIVATE_IP="$(curl -s http://169.254.169.254/latest/meta-data/local-ipv4)"

PUBLIC_IP="$(curl -s http://169.254.169.254/latest/meta-data/public-ipv4)"
                                                          Fetches the public
yum-config-manager --enable epel    Adds extra packages   IP address of the
yum clean all                       to the package        virtual machine
                                    manager yum
yum install -y openswan xl2tpd
                                  Installs software packages
cat > /etc/ipsec.conf <<EOF
[...]                           Writes a configuration
EOF                             file for IPSec (OpenSwan)

cat > /etc/ipsec.secrets <<EOF          Writes a file containing the
$PUBLIC_IP %any : PSK "${IPSEC_PSK}"    shared secret for IPSec
EOF

cat > /etc/xl2tpd/xl2tpd.conf <<EOF       Writes a configuration file
[...]                                     for the L2TP tunnel
EOF

cat > /etc/ppp/options.xl2tpd <<EOF       Writes a configuration file
[...]                                     for the PPP service
EOF

service ipsec start            Starts the services needed
service xl2tpd start           for the VPN server

chkconfig ipsec on           Configures the run level
chkconfig xl2tpd on          for the VPN services
```

That's it. You've now deployed a VPN server to a virtual machine with the help of EC2 user data and a shell script. If you want to test the VPN server, select the VPN type L2TP over IPSec in your VPN client. After you terminate your virtual machine, you'll be ready to learn how to deploy a common web application without writing a custom script.

WARNING You've reached the end of the VPN server example. Don't forget to terminate your virtual machine and clean up your environment. To do so, enter `aws cloudformation delete-stack --stack-name vpn` at your terminal.

5.3.3 Starting from scratch instead of updating

You learned how to deploy an application with the help of user data in this section. The script from the user data is executed at the end of the boot process. But how do you update your application using this approach?

You've automated the installation and configuration of software during your VM's boot process, and can start a new VM without any extra effort. So if you have to update your application or its dependencies, it is easier to create a new up-to-date VM by following these steps:

1. Make sure an up-to-date version of your application or software is available through the package repository of your OS, or edit the user data script.
2. Start a new virtual machine based on your CloudFormation template and user data script.
3. Test the application deployed to the new virtual machine. Proceed with the next step if everything works as it should.
4. Switch your workload to the new virtual machine (for example, by updating a DNS record).
5. Terminate the old virtual machine, and throw away its unused dependencies.

5.4 Deploying a simple web application with AWS Elastic Beanstalk

If you have to deploy a common web application, you don't have to reinvent the wheel. AWS offers a service called *AWS Elastic Beanstalk* that can help you deploy web applications based on Go, Java (SE or Tomcat), .NET on Windows Server with IIS, Node.js, PHP, Python, Ruby, and Docker. With AWS Elastic Beanstalk, you don't have to worry about your OS or virtual machines. AWS will manage them for you (if you enable automatic updates). With Elastic Beanstalk, you only deal with your application. The OS and the runtime (such as Apache + Tomcat) are managed by AWS.

AWS Elastic Beanstalk lets you handle the following recurring problems:

- Providing a runtime environment for a web application (PHP, Java, and so on)
- Updating the runtime environment for a web application
- Installing and updating a web application automatically
- Configuring a web application and its environment
- Scaling a web application to balance load
- Monitoring and debugging a web application

5.4.1 Components of AWS Elastic Beanstalk

Getting to know the different components of AWS Elastic Beanstalk will help you to understand its functionality. Figure 5.4 shows these elements:

- An *application* is a logical container. It contains versions, environments, and configurations. If you start to use AWS Elastic Beanstalk in a region, you have to create an application first.
- A *version* contains a specific release of your application. To create a new version, you have to upload your executables (packed into an archive) to Amazon S3, which stores static files. A version is basically a pointer to this archive of executables.
- A *configuration template* contains your default configuration. You can manage your application's configuration (such as the port your application listens on) as well as the environment's configuration (such as the size of the virtual machine) with your custom configuration template.
- An *environment* is where AWS Elastic Beanstalk executes your application. It consists of a *version* and the *configuration*. You can run multiple environments for one application by using different combinations of versions and configurations.

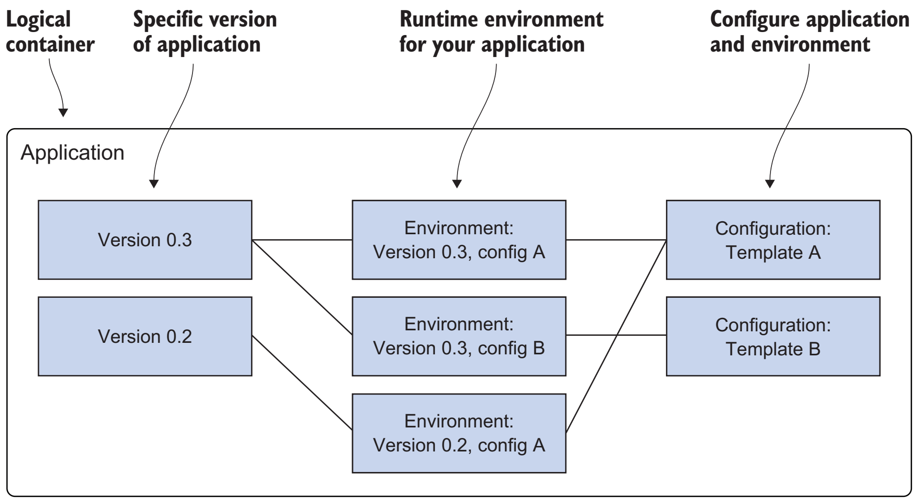

Logical container
Specific version of application
Runtime environment for your application
Configure application and environment
Application
Version 0.3
Version 0.2
Environment: Version 0.3, config A
Environment: Version 0.3, config B
Environment: Version 0.2, config A
Configuration: Template A
Configuration: Template B

Figure 5.4 An AWS Elastic Beanstalk application consists of versions, environments, and configurations.

Enough theory for the moment. Let's proceed with deploying a simple web application.

5.4.2 Using AWS Elastic Beanstalk to deploy Etherpad, a Node.js application

Editing a document collaboratively can be painful if you're using the wrong tools. *Etherpad* is an open source online editor that lets you edit a document with many people in real time. You'll deploy this Node.js-based application with the help of AWS Elastic Beanstalk in three steps:

1 Create an application: the logical container.
2 Create a version: a pointer to a specific version of Etherpad.
3 Create an environment: the place where Etherpad will run.

Creating an application for AWS Elastic Beanstalk

Open your terminal, and execute the following command to create an application for the AWS Elastic Beanstalk service:

```
$ aws elasticbeanstalk create-application --application-name etherpad
```

You've now created a container for all the other components that are necessary to deploy Etherpad with the help of AWS Elastic Beanstalk.

Creating a version for AWS Elastic Beanstalk

You can create a new version of your Etherpad application with the following command:

```
$ aws elasticbeanstalk create-application-version --application-name etherpad \
➥ --version-label 1 \
➥ --source-bundle "S3Bucket=awsinaction-code2,S3Key=chapter05/etherpad.zip"
```

By executing this command, you created a version labeled `1`. For this example, we uploaded a zip archive containing Etherpad that you can use for convenience.

Creating an environment to execute Etherpad with AWS Elastic Beanstalk

To deploy Etherpad with the help of AWS Elastic Beanstalk, you have to create an environment for Node.js based on Amazon Linux and the version of Etherpad you just created. To get the latest Node.js environment version, called a *solution stack name*, run this command:

```
$ aws elasticbeanstalk list-available-solution-stacks --output text \
➥ --query "SolutionStacks[?contains(@, 'running Node.js')] | [0]"
64bit Amazon Linux 2017.03 v4.2.1 running Node.js
```

When AWS releases a new solution stack, this output may look different.

Execute the following command to launch an environment, replacing *`$SolutionStackName`* with the output from the previous command.

```
$ aws elasticbeanstalk create-environment --environment-name etherpad \
➥ --application-name etherpad \
➥ --option-settings Namespace=aws:elasticbeanstalk:environment,\
➥ OptionName=EnvironmentType,Value=SingleInstance \
➥ --solution-stack-name "$SolutionStackName" \
➥ --version-label 1
```

Launches a single virtual machine without the ability to scale and load-balance automatically

Having fun with Etherpad

You've now created an environment for Etherpad. It will take several minutes before you can point your browser to your Etherpad installation. The following command will help you track the state of your Etherpad environment:

```
$ aws elasticbeanstalk describe-environments --environment-names etherpad
```

When Status turns to Ready and Health turns to Green, you're ready to create your first Etherpad document. The output of the `describe` command should look similar to the following example.

Listing 5.4 Describing the status of the Elastic Beanstalk environment

```
{
  "Environments": [{
    "ApplicationName": "etherpad",
    "EnvironmentName": "etherpad",
    "VersionLabel": "1",
    "Status": "Ready",                    <- Wait until Status turns to Ready.
    "EnvironmentLinks": [],
    "PlatformArn": "arn:aws:elasticbeanstalk:us-east-1::platform/Node.js
➥ running on 64bit Amazon Linux/4.2.1",
    "EndpointURL": "54.157.76.149",
    "SolutionStackName": "64bit Amazon Linux 2017.03 v4.2.1 running Node.js",
    "EnvironmentId": "e-8d532q3vkk",
    "CNAME": "etherpad.d2nhjs7myw.us-east-1.elasticbeanstalk.com",   <- DNS record for the environment (for example, to open with a browser)
    "AbortableOperationInProgress": false,
    "Tier": {
      "Version": " ",
      "Type": "Standard",
      "Name": "WebServer"
    },
    "Health": "Green",                    <- Wait until Health turns to Green.
    "DateUpdated": "2017-08-15T09:18:47.750Z",
    "DateCreated": "2017-08-15T09:14:32.137Z"
  }]
}
```

You've now deployed a Node.js web application to AWS in three simple steps. Point your browser to the URL shown in `CNAME`, and open a new document by typing in a name for it and clicking OK. If the page does not load, try the `EndpointURL`, which is a public IP address. The CNAME should work within the next few minutes as well. Figure 5.5 shows an Etherpad document in action.

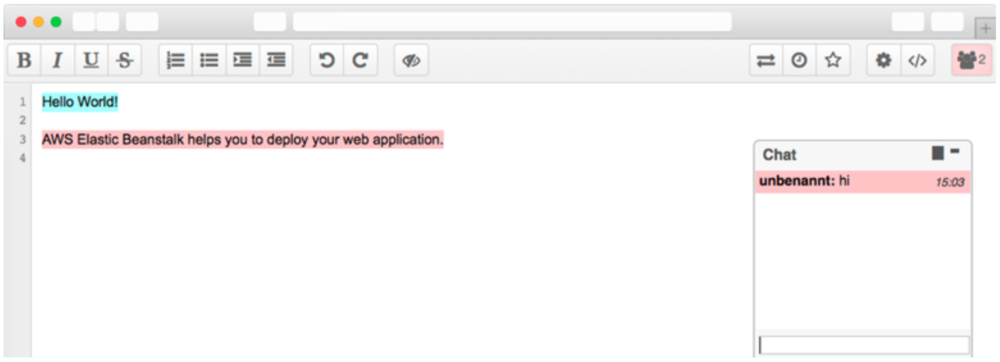

Hello World!
AWS Elastic Beanstalk helps you to deploy your web application.
Chat
unbenannt: hi
15:03

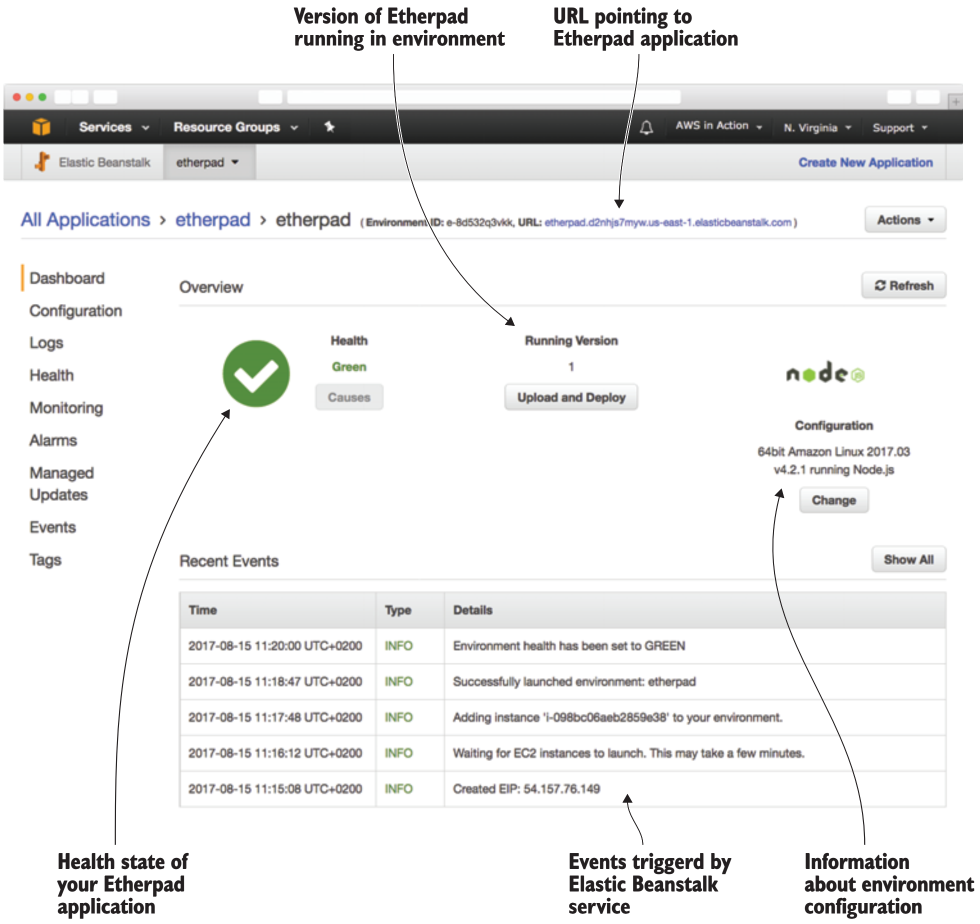
Figure 5.5 Online text editor Etherpad in action

If you want to deploy any other Node.js application, the only thing that changes is the zip file that you upload to Elastic Beanstalk. If you want to run something other than a Node.js application, you have to use the appropriate solution stack name with `aws elasticbeanstalk list-available-solution-stacks`.

Exploring AWS Elastic Beanstalk with the Management Console

You've deployed Etherpad using AWS Elastic Beanstalk and the AWS CLI by creating an application, a version, and an environment. You can also control AWS Elastic Beanstalk using the web-based Management Console. In our experience, the Management Console is the best way to manage AWS Elastic Beanstalk.

1. Open the AWS Management Console at https://console.aws.amazon.com.
2. Click Services in the navigation bar, and click the Elastic Beanstalk service.
3. Click the etherpad environment, represented by a green box. An overview of the Etherpad application is shown, as in figure 5.6.



(Environment ID: e-8d532q3vkk, URL: etherpad.d2nhjs7myw.us-east-1.elasticbeanstalk.com)

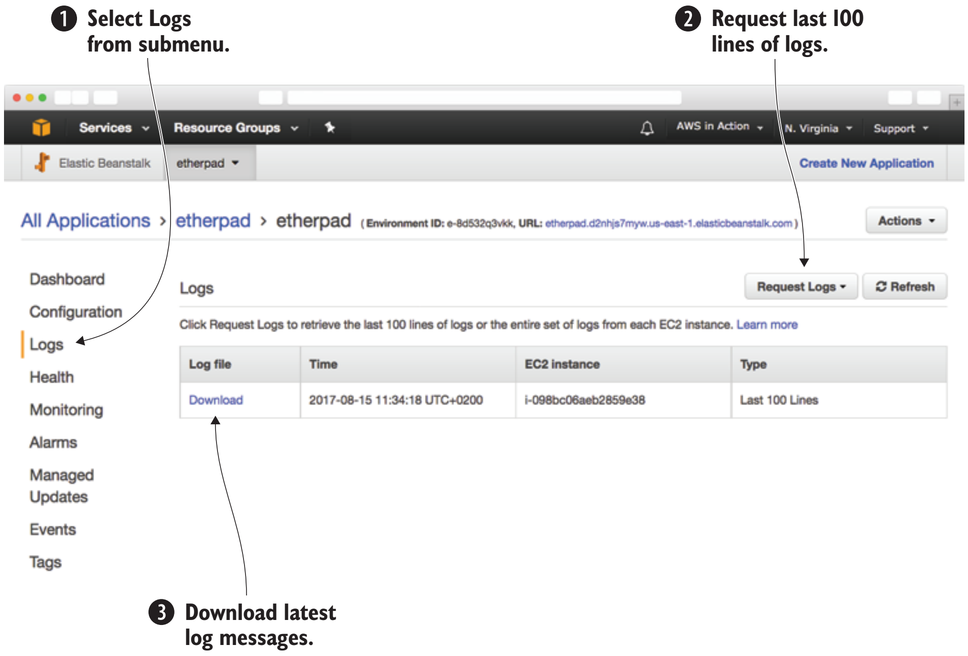
64bit Amazon Linux 2017.03 v4.2.1 running Node.js

Time	Type	Details
2017-08-15 11:20:00 UTC+0200	INFO	Environment health has been set to GREEN
2017-08-15 11:18:47 UTC+0200	INFO	Successfully launched environment: etherpad
2017-08-15 11:17:48 UTC+0200	INFO	Adding instance 'i-098bc06aeb2859e38' to your environment.
2017-08-15 11:16:12 UTC+0200	INFO	Waiting for EC2 instances to launch. This may take a few minutes.
2017-08-15 11:15:08 UTC+0200	INFO	Created EIP: 54.157.76.149

Figure 5.6 Overview of AWS Elastic Beanstalk environment running Etherpad

What if something goes wrong with your application? How can you debug an issue? Usually you connect to the virtual machine and look at the log messages. You can fetch the log messages from your application (and other components) using AWS Elastic Beanstalk. Follow these steps:

1 Choose Logs from the submenu. You'll see a screen like that shown in figure 5.7.
2 Click Request Logs, and choose Last 100 Lines.
3 After a few seconds, a new entry will appear in the table. Click Download to download the log file to your computer.



All Applications › etherpad › etherpad (Environment ID: e-8d532q3vkk, URL: etherpad.d2nhjs7myw.us-east-1.elasticbeanstalk.com)

Logs

Click Request Logs to retrieve the last 100 lines of logs or the entire set of logs from each EC2 instance. Learn more

Log file	Time	EC2 instance	Type
Download	2017-08-15 11:34:18 UTC+0200	i-098bc06aeb2859e38	Last 100 Lines

Figure 5.7 Downloading logs from a Node.js application via AWS Elastic Beanstalk

Cleaning up AWS Elastic Beanstalk

Now that you've successfully deployed Etherpad using AWS Elastic Beanstalk and learned about the service's different components, it's time to clean up. Run the following command to terminate the Etherpad environment:

```
$ aws elasticbeanstalk terminate-environment --environment-name etherpad
```

You can check the state of the environment by executing the following command:

```
$ aws elasticbeanstalk describe-environments --environment-names etherpad \
➥ --output text --query "Environments[].Status"
```

Wait until Status has changed to Terminated, and then proceed with the following command:

```
$ aws elasticbeanstalk delete-application --application-name etherpad
```

That's it. You've terminated the virtual machine providing the environment for Etherpad and deleted all components of AWS Elastic Beanstalk.

5.5 Deploying a multilayer application with AWS OpsWorks Stacks

Deploying a basic web application using AWS Elastic Beanstalk is convenient. But if you have to deploy a more complex application consisting of different services—also called *layers*—you'll reach the limits of AWS Elastic Beanstalk. In this section, you'll learn about AWS OpsWorks Stacks, a free service offered by AWS that can help you to deploy a multilayer application.

AWS OpsWorks flavors

AWS OpsWorks comes in different two flavors:

- *AWS OpsWorks Stacks* comes with Chef versions 11 and 12. In Chef 11, OpsWorks comes with a bunch of built-in layers, which is best for beginners. If you have Chef knowledge, this may limit you. We recommend to use OpsWorks Stacks with Chef 12 if you have Chef knowledge, because there are no limiting built-in layers.
- *AWS OpsWorks for Chef Automate* provides a Chef Automate server and takes care about backups, restorations, and software upgrades. You should use OpsWorks for Chef Automate if you have an existing infrastructure managed by Chef that you want to migrate to AWS.

AWS OpsWorks Stacks helps you control AWS resources like virtual machines, load balancers, container clusters, and databases, and lets you deploy applications. The service offers some standard layers with the following runtimes:

- HAProxy (load balancer)
- Static web server
- Rails app server (Ruby on Rails)
- PHP app server
- Node.js app server
- Java app server (Tomcat server)
- AWS Flow (Ruby)
- MySQL (database)
- Memcached (in-memory cache)
- Ganglia (monitoring)

You can also add a custom layer to deploy anything you want. The deployment is controlled with the help of *Chef*, a configuration management tool. Chef uses *recipes* organized into *cookbooks* to deploy applications to any kind of system. You can adopt the standard recipes or create your own.

About Chef

Chef is a configuration management tool similar to Puppet, SaltStack, CFEngine, and Ansible. Chef lets you configure and deploy applications by transforming templates (recipes) written in a domain-specific language (DSL) into actions. A recipe can include packages to install, services to run, or configuration files to write, for example. Related recipes can be combined into cookbooks. Chef analyzes the status quo and changes resources where necessary to reach the described state from the recipe.

You can reuse cookbooks and use recipes you get from others. The community publishes a variety of cookbooks and recipes at https://supermarket.chef.io under open source licenses.

Chef can be run in solo or client/server mode. It acts as a fleet-management tool in client/server mode. This can help if you have to manage a distributed system consisting of many VMs. In solo mode, you can execute recipes on a single VM. AWS OpsWorks uses solo mode integrated into its own fleet management, without requiring you to configure and operate a setup in client/server mode.

Besides deploying your application, AWS OpsWorks Stacks can help you to scale, monitor, and update your VMs running beneath the different layers.

5.5.1 Components of AWS OpsWorks Stacks

Getting to know the different components of AWS OpsWorks Stacks will help you understand its functionality. Figure 5.8 shows these elements:

- A *stack* is a container for all other components of AWS OpsWorks Stacks. You can create one or more stacks and add one or more layers to each stack. You could use different stacks to separate the production environment from the testing environment, for example. Or you could use different stacks to separate different applications.
- A *layer* belongs to a stack. A layer represents an application; you could also call it a service. AWS OpsWorks Stacks offers predefined layers for standard web applications like PHP and Java, but you're free to use a custom stack for any application you can think of. Layers are responsible for configuring and deploying software to virtual machines. You can add one or multiple VMs to a layer; in this context the VMs are called *instances*.
- An *instance* is the representation for a virtual machine. You can launch one or multiple instances for each layer, using different versions of Amazon Linux and

Ubuntu or a custom AMI as a basis for the instances. You can specify rules for launching and terminating instances based on load or timeframes for scaling.

- An *app* is the software you want to deploy. AWS OpsWorks Stacks deploys your app to a suitable layer automatically. You can fetch apps from a Git or Subversion repository, or as archives via HTTP. AWS OpsWorks Stacks helps you to install and update your apps onto one or multiple instances.

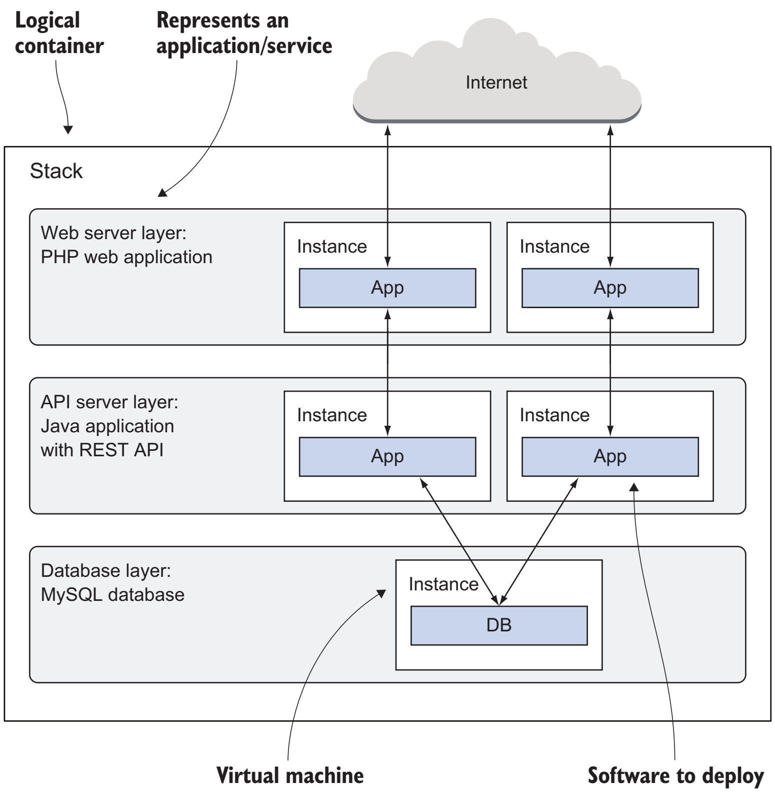

Logical container
Represents an application/service
Internet
Stack
Web server layer: PHP web application
Instance
App
Instance
App
API server layer: Java application with REST API
Instance
App
Instance
App
Database layer: MySQL database
Instance
DB
Virtual machine
Software to deploy

Figure 5.8 Stacks, layers, instances, and apps are the main components of AWS OpsWorks Stacks.

Let's look at how to deploy a multilayer application with the help of AWS OpsWorks Stacks.

5.5.2 Using AWS OpsWorks Stacks to deploy an IRC chat application

Internet Relay Chat (IRC) is still a popular means of communication in some circles. In this section, you'll deploy *kiwiIRC*, a web-based IRC client, and your own IRC server. Figure 5.9 shows the setup of a distributed system consisting of a web application delivering the IRC client, as well as an IRC server.

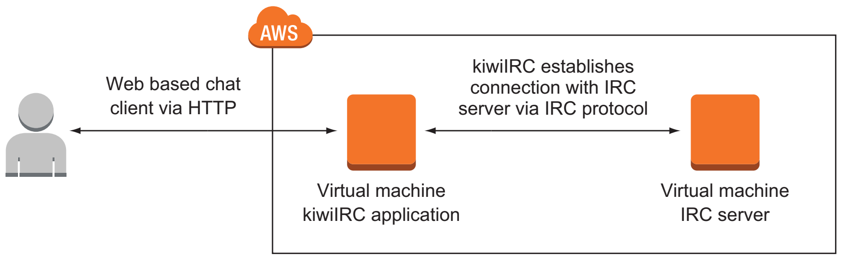

AWS
Web based chat client via HTTP
kiwiIRC establishes connection with IRC server via IRC protocol
Virtual machine kiwiIRC application
Virtual machine IRC server

Figure 5.9 Building your own IRC infrastructure consisting of a web application and an IRC server

kiwiIRC is an open source web application written in JavaScript for Node.js. To deploy it as a two-layer application using AWS OpsWorks Stacks, you need scripts that do the following:

1. Create a stack, the container for all other components.
2. Create a Node.js layer for kiwiIRC.
3. Create a custom layer for the IRC server.
4. Create an app to deploy kiwiIRC to the Node.js layer.
5. Add an instance for each layer.

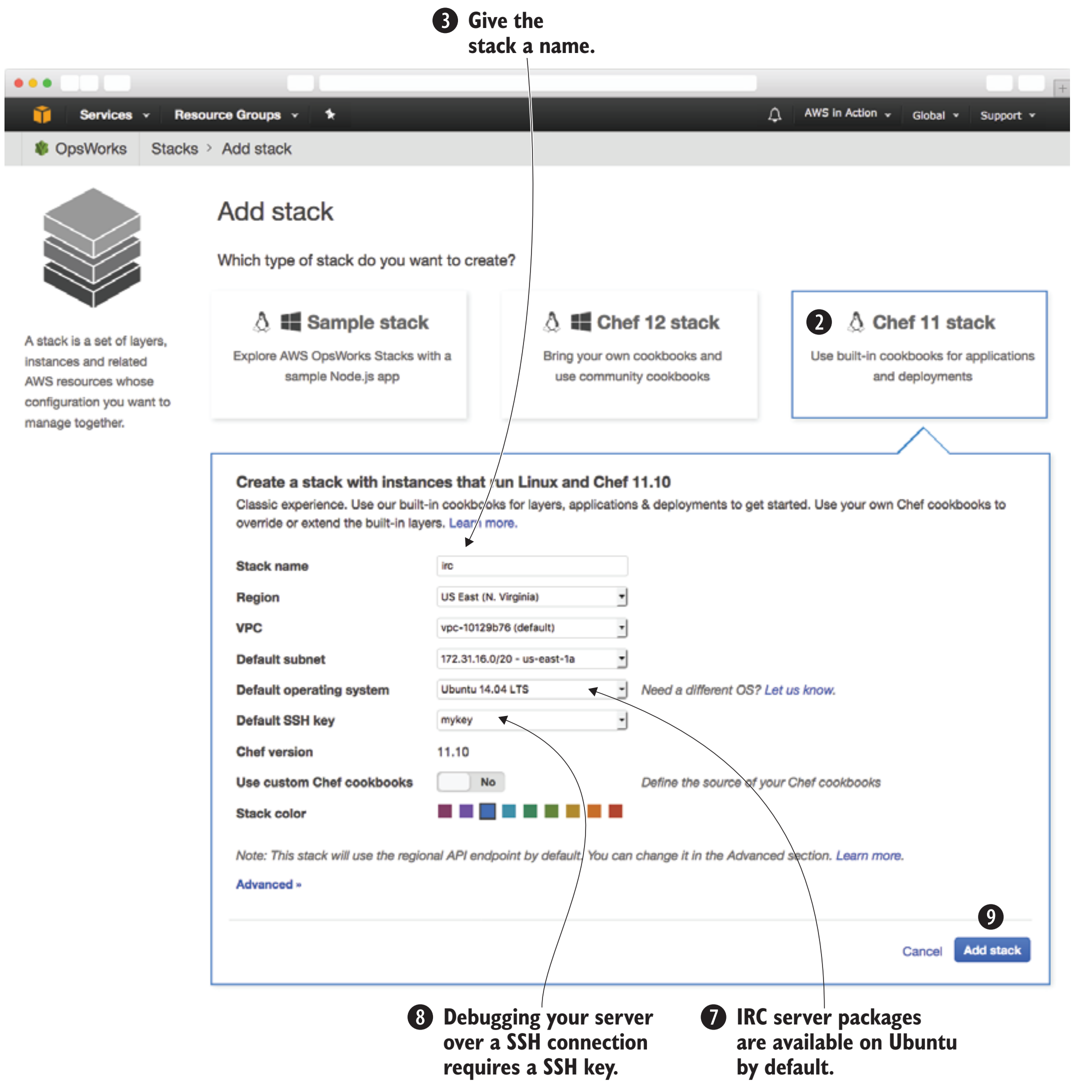
You'll learn how to handle these steps with the Management Console. You can also control AWS OpsWorks Stacks with the CLI, just like AWS Elastic Beanstalk or AWS CloudFormation.

CREATING A NEW OPSWORKS STACK

Open the Management Console at https://console.aws.amazon.com/opsworks, and click the Go to OpsWorks Stacks button. There you can start fresh by adding a new stack. Figure 5.10 illustrates the necessary steps with the most important highlighted:

1. Click Add stack under Select Stack or Add Your First Stack.
2. Select Chef 11 stack.
3. For Name, type in `irc`.
4. For Region, choose US East (N. Virginia).
5. The default VPC is the only one available. Select it.
6. For Default subnet, select us-east-1a.
7. For Default operating system, choose Ubuntu 14.04 LTS.
8. Select your SSH key, mykey, for Default SSH key.
9. Click Add stack to create the stack.

You're now redirected to an overview of your IRC AWS OpsWorks stack. Everything is ready for you to create the first layer.


3 Give the stack a name.
Services
Resource Groups
AWS in Action
Global
Support
OpsWorks
Stacks > Add stack
2
9
8 Debugging your server over a SSH connection requires a SSH key.
7 IRC server packages are available on Ubuntu by default.

Add stack

Which type of stack do you want to create?

A stack is a set of layers, instances and related AWS resources whose configuration you want to manage together.

Sample stack Explore AWS OpsWorks Stacks with a sample Node.js app

Chef 12 stack Bring your own cookbooks and use community cookbooks

Chef 11 stack Use built-in cookbooks for applications and deployments

Create a stack with instances that run Linux and Chef 11.10

Classic experience. Use our built-in cookbooks for layers, applications & deployments to get started. Use your own Chef cookbooks to override or extend the built-in layers. Learn more.

Stack name	irc	
Region	US East (N. Virginia)	
VPC	vpc-10129b76 (default)	
Default subnet	172.31.16.0/20 - us-east-1a	
Default operating system	Ubuntu 14.04 LTS	*Need a different OS? Let us know.*
Default SSH key	mykey	
Chef version	11.10	
Use custom Chef cookbooks	No	*Define the source of your Chef cookbooks*
Stack color	🟪 🟪 🟦 🟦 🟩 🟩 🟨 🟧 🟥	

Note: This stack will use the regional API endpoint by default. You can change it in the Advanced section. Learn more.

Advanced »

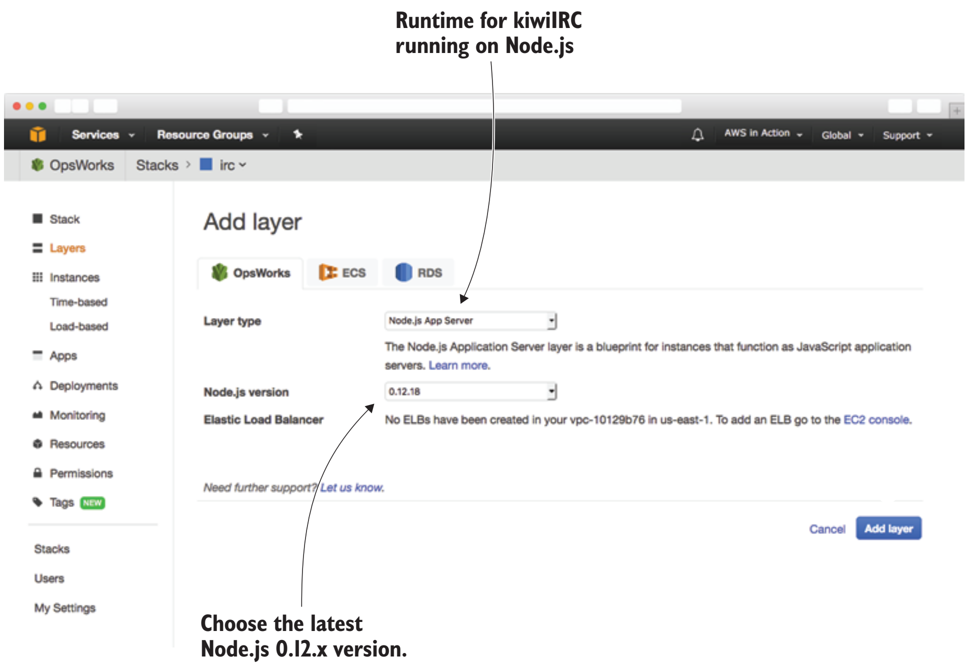
Cancel Add stack

Figure 5.10 Creating a stack with AWS OpsWorks Stacks

Creating a Node.js layer for an OpsWorks stack

kiwiIRC is a Node.js application, so you need to create a Node.js layer for the IRC stack. Follow these steps in figure 5.11 to do so:

1. Select Layers from the submenu on the left.
2. Click the Add layer button.
3. For Layer type, select Node.js App Server.
4. Select the latest 0.12.x version of Node.js.
5. Click Add layer.



Add layer

Layer type: Node.js App Server

The Node.js Application Server layer is a blueprint for instances that function as JavaScript application servers. Learn more.

Node.js version: 0.12.18

Elastic Load Balancer: No ELBs have been created in your vpc-10129b76 in us-east-1. To add an ELB go to the EC2 console.

Need further support? Let us know.

Figure 5.11 Creating a layer with Node.js for kiwiIRC

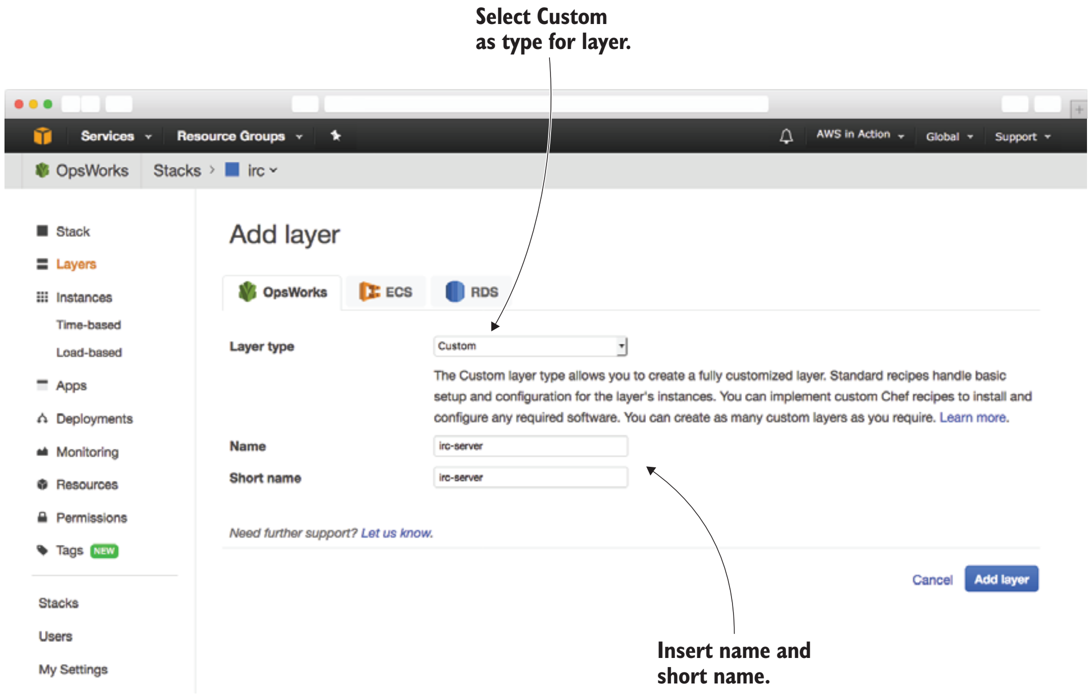
You've created a Node.js layer. Now you need to repeat these steps to add another layer and deploy your own IRC server.

CREATING A CUSTOM LAYER FOR AN OPSWORKS STACK

An IRC server isn't a typical web application, so the default layer types are out of the question. You'll use a custom layer for deploying an IRC server. The Ubuntu package repository includes various IRC server implementations; you'll use the `ircd-ircu` package. Follow these steps in figure 5.12 to create a custom stack for the IRC server:

1. Select Layers from the submenu on the left.
2. Click Add layer.
3. For Layer type, select Custom.
4. For Name and for Short name, type in `irc-server`.
5. Click Add layer.

You've now created a custom layer. If you want to deploy any other application, go with one of the pre-built layers first. If that is not possible, use a custom layer. This way, you benefit best from OpsWorks.



Layer type: Custom

The Custom layer type allows you to create a fully customized layer. Standard recipes handle basic setup and configuration for the layer's instances. You can implement custom Chef recipes to install and configure any required software. You can create as many custom layers as you require. Learn more.

Name: irc-server

Short name: irc-server

Need further support? Let us know.

Figure 5.12 Creating a custom layer to deploy an IRC server

The IRC server needs to be reachable through port 6667. To allow access to this port, you need to define a custom firewall. Execute the commands shown in listing 5.5 to create a custom firewall for your IRC server.

Shortcut for Linux and macOS

You can avoid typing these commands manually into your terminal by using the following command to download a Bash script and execute it directly on your local machine. The Bash script contains the same steps as shown in listing 5.5:

```
$ curl -s https://raw.githubusercontent.com/AWSinAction/\
➥ code2/master/chapter05/irc-create-cloudformation-stack.sh \
➥ | bash -ex
```

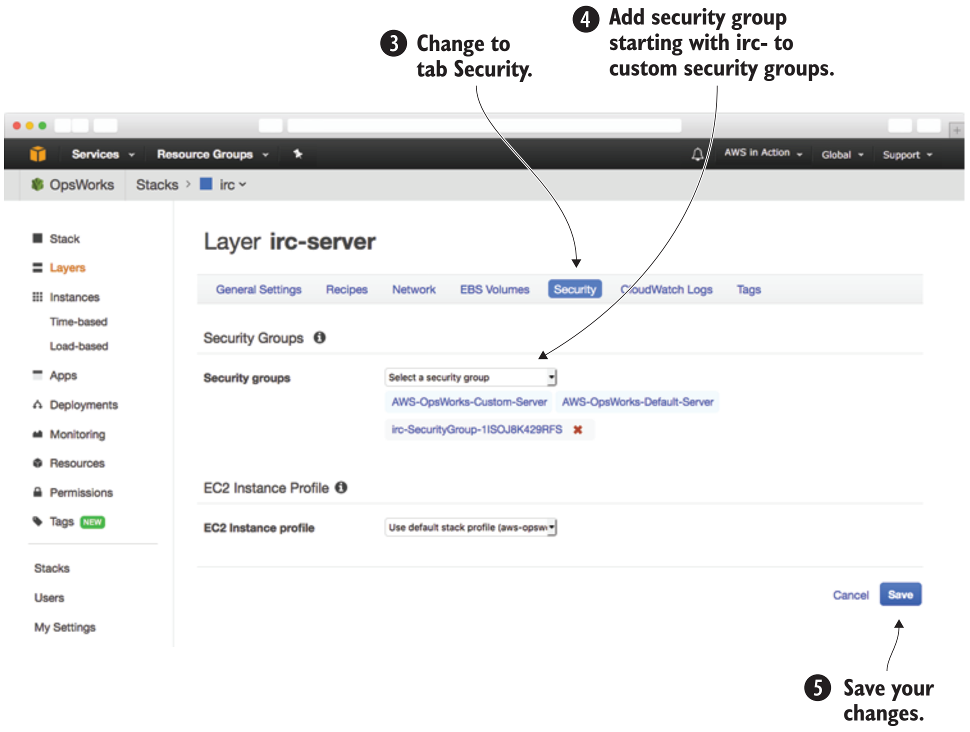
Listing 5.5 Creating a custom firewall with the help of CloudFormation

```
$ VpcId="$(aws ec2 describe-vpcs --query "Vpcs[0].VpcId" --output text)"    ◁ Gets the default VPC

$ aws cloudformation create-stack --stack-name irc \    ◁ Creates a CloudFormation stack
➥ --template-url https://s3.amazonaws.com/awsinaction-code2/\
➥ chapter05/irc-cloudformation.yaml \
➥ --parameters "ParameterKey=VPC,ParameterValue=$VpcId"
$ aws cloudformation wait stack-create-complete --stack-name irc
```
Wait until stack is CREATE_COMPLETE.

Next you need to attach this custom firewall configuration to the custom OpsWorks layer. Follow these steps in figure 5.13:

1. Select Layers from the submenu on the left.
2. Open the irc-server layer by clicking it.
3. Change to the Security tab, and click Edit.
4. For custom Security groups, select the security group that starts with irc.
5. Click Save.



Layer irc-server

Security Groups

Security groups: Select a security group
AWS-OpsWorks-Custom-Server AWS-OpsWorks-Default-Server
irc-SecurityGroup-1ISOJ8K429RFS

EC2 Instance Profile

EC2 Instance profile: Use default stack profile (aws-opsw

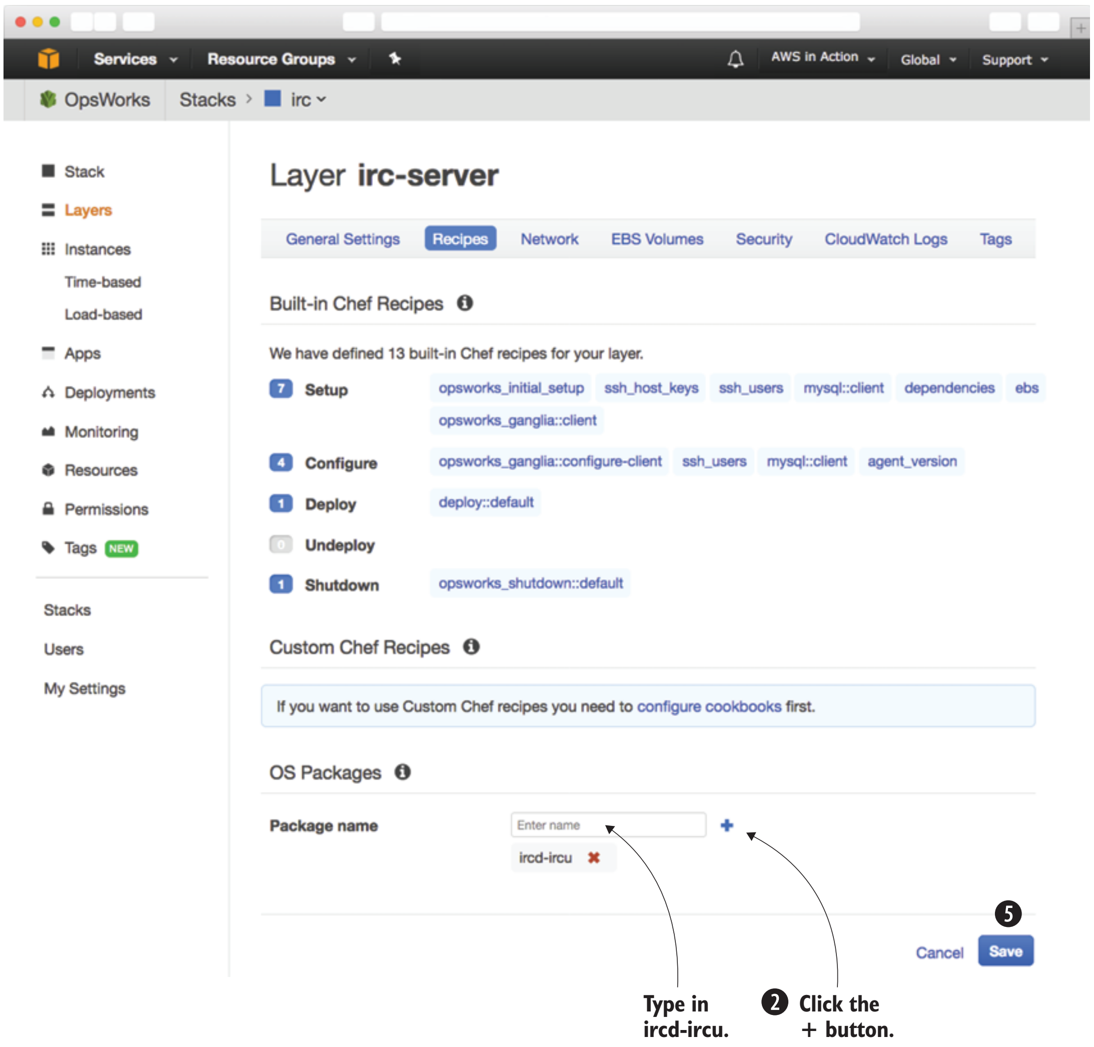
Figure 5.13 Adding a custom firewall configuration to the IRC server layer

You need to configure one last thing for the IRC server layer: the layer recipes for deploying an IRC server. Follow these steps in figure 5.14 to do so:

1. Select Layers from the submenu on the left.
2. Open the irc-server layer by clicking it.
3. Change to the Recipes tab, and click Edit.

4 For OS Packages, add the package `ircd-ircu`. Don't forget to click the + button to add the package.
5 Click save.

You've successfully created and configured a custom layer to deploy the IRC server. Next you'll add the kiwiIRC web application as an app to OpsWorks.



Layer **irc-server**

Built-in Chef Recipes

We have defined 13 built-in Chef recipes for your layer.

7 **Setup**	opsworks_initial_setup, ssh_host_keys, ssh_users, mysql::client, dependencies, ebs, opsworks_ganglia::client
4 **Configure**	opsworks_ganglia::configure-client, ssh_users, mysql::client, agent_version
1 **Deploy**	deploy::default
0 **Undeploy**	
1 **Shutdown**	opsworks_shutdown::default

Custom Chef Recipes

If you want to use Custom Chef recipes you need to configure cookbooks first.

OS Packages

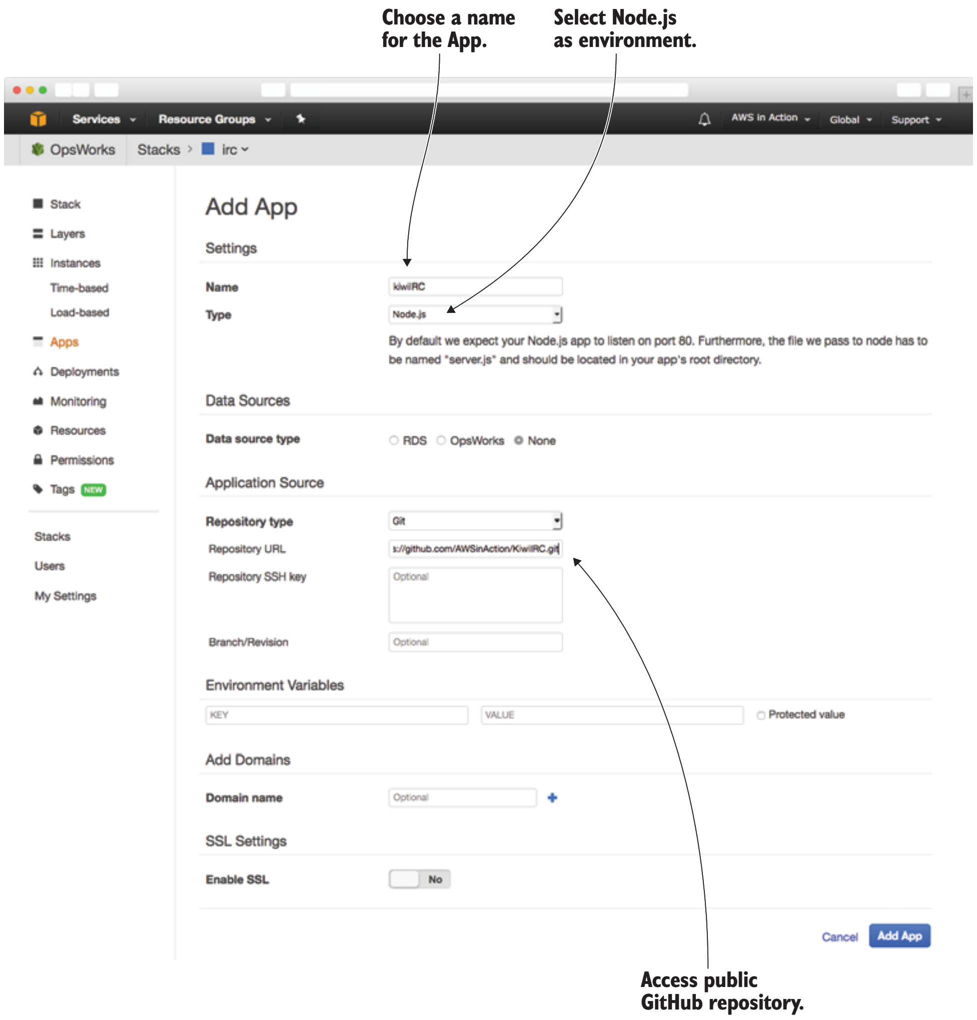
Package name Enter name +
ircd-ircu ✖

Figure 5.14 Adding an IRC package to a custom layer

ADDING AN APP TO THE NODE.JS LAYER

Now you're ready to deploy an app to the Node.js layer you just created. Follow these steps in figure 5.15:

1. Select Apps from the submenu.
2. Click the Add app button.
3. For Name, type in `kiwiIRC`.
4. For Type, select Node.js.
5. For Repository type, select Git, and type in `https://github.com/AWSinAction/KiwiIRC.git` for Repository URL.
6. Click the Add App button.



Add App

Settings

Name: kiwiIRC
Type: Node.js

By default we expect your Node.js app to listen on port 80. Furthermore, the file we pass to node has to be named "server.js" and should be located in your app's root directory.

Data Sources

Data source type: RDS OpsWorks [✓ None]

Application Source

Repository type: Git
Repository URL: s://github.com/AWSinAction/KiwiIRC.git
Repository SSH key: Optional
Branch/Revision: Optional

Environment Variables

KEY VALUE Protected value

Add Domains

Domain name: Optional

SSL Settings

Enable SSL: No

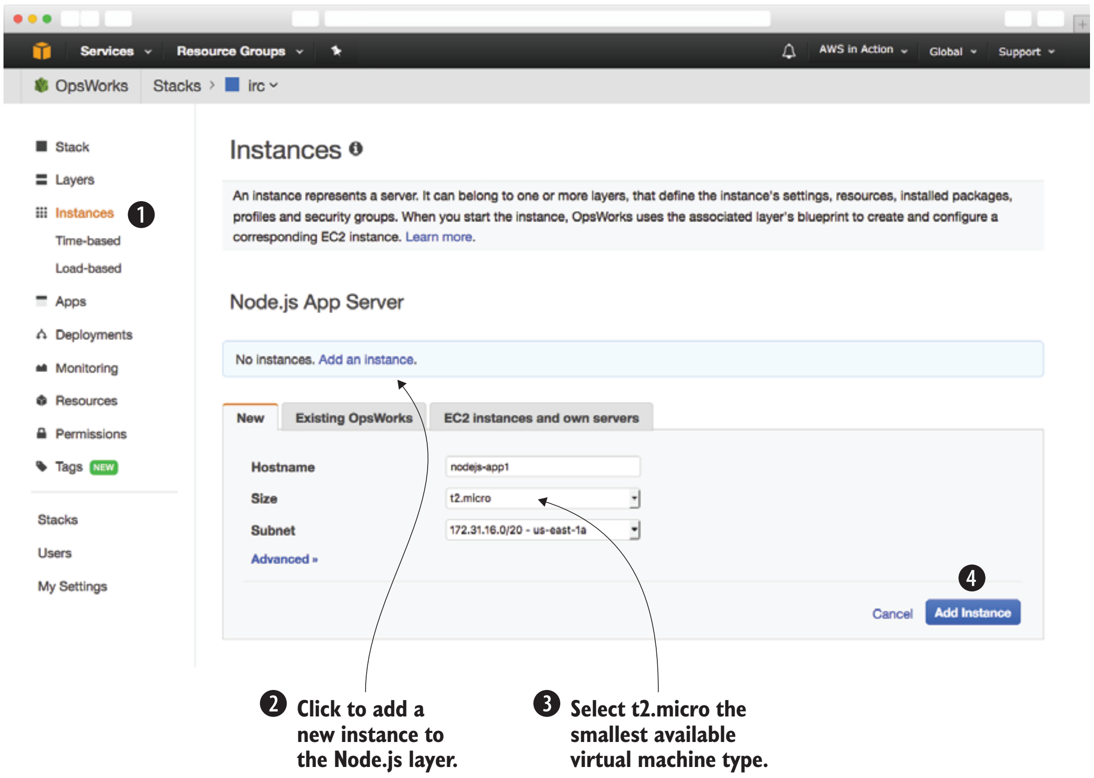
Figure 5.15 Adding kiwiIRC, a Node.js app, to OpsWorks

Your first OpsWorks stack is now fully configured. Only one thing is missing: you need to start some instances.

Adding instances to run the IRC client and server

Adding two instances will bring the kiwiIRC client and the IRC server into being. Adding a new instance to a layer is easy—follow these steps shown in figure 5.16:

1. Select Instances from the submenu on the left.
2. Click the Add instance button on the Node.js App Server layer.
3. For Size, select t2.micro, the instance type covered by the Free Tier.
4. Click Add instance.



Instances

An instance represents a server. It can belong to one or more layers, that define the instance's settings, resources, installed packages, profiles and security groups. When you start the instance, OpsWorks uses the associated layer's blueprint to create and configure a corresponding EC2 instance. Learn more.

Node.js App Server

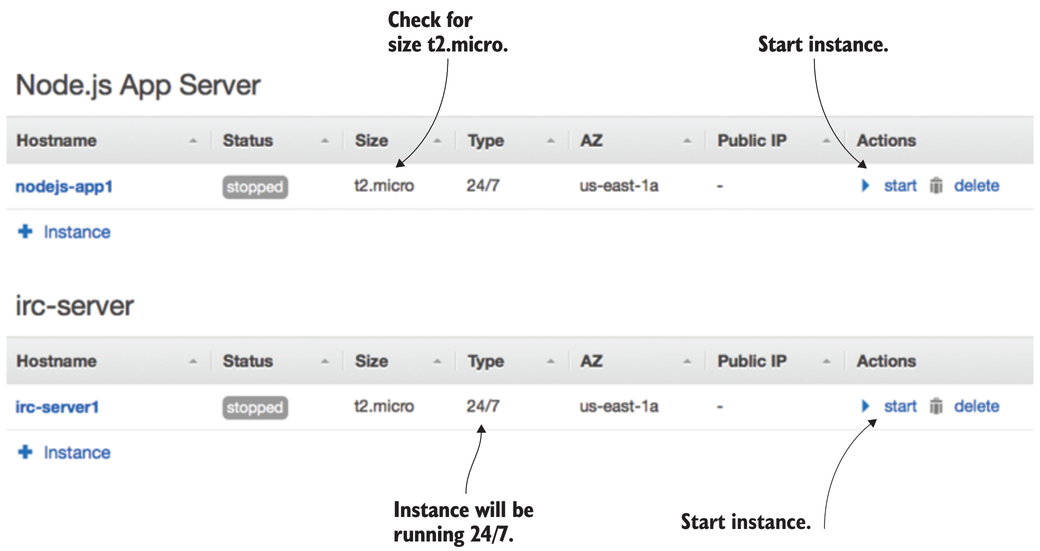
No instances. Add an instance.

Hostname	nodejs-app1
Size	t2.micro
Subnet	172.31.16.0/20 - us-east-1a

Figure 5.16 Adding a new instance to the Node.js layer

You've added an instance to the Node.js App Server layer. Repeat these steps for the irc-server layer as well.

The overview of instances should be similar to figure 5.17. To start them, click Start for both instances. It will take some time for the virtual machines to boot and the deployment to run, so this is a good time to get some coffee or tea.



Node.js App Server

Hostname	Status	Size	Type	AZ	Public IP	Actions
nodejs-app1	stopped	t2.micro	24/7	us-east-1a	-	start delete

+ Instance

irc-server

Hostname	Status	Size	Type	AZ	Public IP	Actions
irc-server1	stopped	t2.micro	24/7	us-east-1a	-	start delete

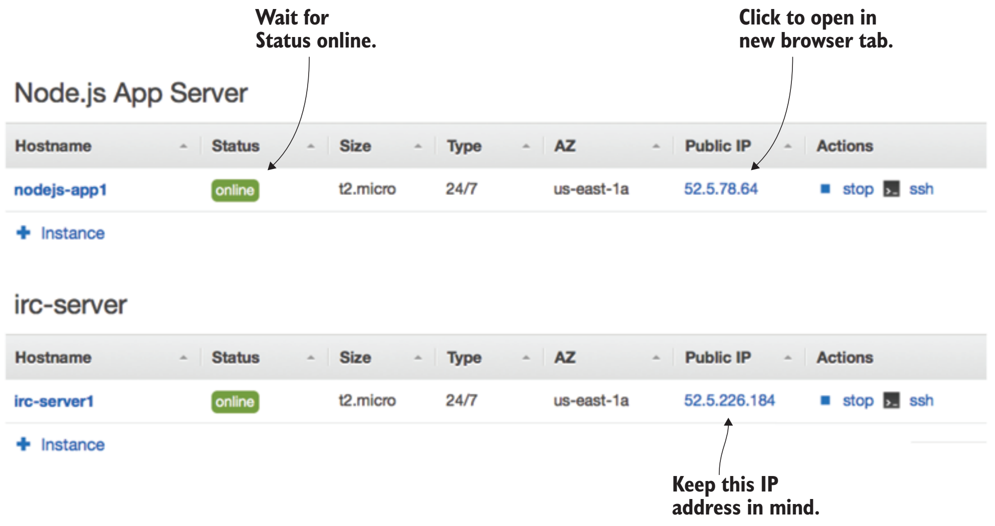
+ Instance

Figure 5.17 Starting the instances for the IRC web client and server

HAVING FUN WITH KIWIIRC

Be patient and wait until the status of both instances changes to Online, as shown in figure 5.18. You can now open kiwiIRC in your browser by following these steps:

1. Remember (or write down) the public IP address of the instance irc-server1. You'll need it to connect to your IRC server later.
2. Click the public IP address of the nodejs-app1 instance to open kiwiIRC in a new browser tab.


Wait for Status online.
Click to open in new browser tab.
Keep this IP address in mind.

Node.js App Server

Hostname	Status	Size	Type	AZ	Public IP	Actions
nodejs-app1	online	t2.micro	24/7	us-east-1a	52.5.78.64	stop ssh

+ Instance

irc-server

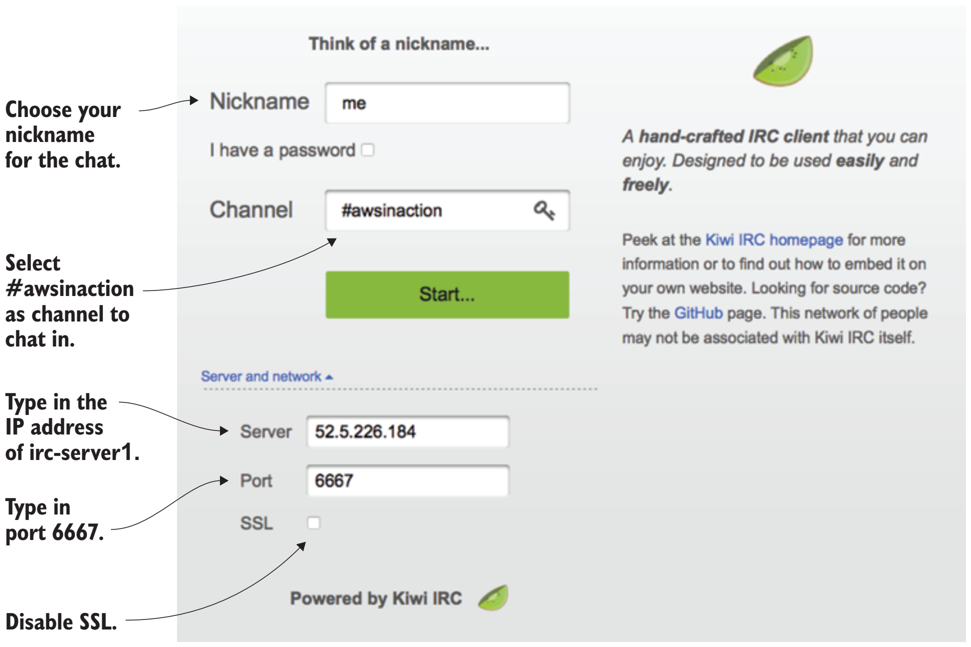
Hostname	Status	Size	Type	AZ	Public IP	Actions
irc-server1	online	t2.micro	24/7	us-east-1a	52.5.226.184	stop ssh

+ Instance

Figure 5.18 Waiting for deployment to open kiwiIRC in the browser

The kiwiIRC application should load in your browser, and you should see a login screen like the one in figure 5.19. Follow these steps to log in to your IRC server with the kiwiIRC web client:

1. Type in a nickname.
2. For Channel, type in `#awsinaction`.
3. Open the details of the connection by clicking Server and network.
4. Type the IP address of irc-server1 into the Server field.
5. For Port, type in `6667`.
6. Disable SSL.
7. Click Start, and wait a few seconds.


Choose your nickname for the chat.
Select #awsinaction as channel to chat in.
Type in the IP address of irc-server1.
Type in port 6667.
Disable SSL.
Start...
Server and network ▴
Powered by Kiwi IRC

Think of a nickname...

Nickname: me

I have a password ☐

Channel: #awsinaction

*A **hand-crafted IRC client** that you can enjoy. Designed to be used **easily** and **freely**.*

Peek at the Kiwi IRC homepage for more information or to find out how to embed it on your own website. Looking for source code? Try the GitHub page. This network of people may not be associated with Kiwi IRC itself.

Server: 52.5.226.184

Port: 6667

SSL: ☐

Figure 5.19 Using kiwiIRC to log in to your IRC server on channel #awsinaction

Congratulations! You've deployed a web-based IRC client and an IRC server with the help of AWS OpsWorks.

Cleaning up AWS OpsWorks

It's time to clean up. Follow these steps to avoid being charged unintentionally:

1. Open the AWS OpsWorks Stacks service with the Management Console.
2. Select the irc stack by clicking it.
3. Select Instances from the submenu..

(continued)

4. Delete both instances, and wait until they disappear from the overview.
5. Select Apps from the submenu.
6. Delete the kiwiIRC app.
7. Select Stack from the submenu.
8. Click the Delete Stack button, and confirm the deletion.
9. Execute `aws cloudformation delete-stack --stack-name irc` from your terminal.

Summary

- Automating the deployment of your applications onto virtual machines allows you to take full advantage of the cloud: scalability and high availability.
- AWS offers different tools for deploying applications onto virtual machines. Using one of these tools prevents you from reinventing the wheel.
- You can update an application by throwing away old VMs and starting new, up-to-date ones if you've automated your deployment process.
- Injecting Bash or PowerShell scripts into a virtual machine during startup allows you to initialize virtual machines individually—for example for installing software or configuring services.
- AWS OpsWorks is good for deploying multilayer applications with the help of Chef.
- AWS Elastic Beanstalk is best suited for deploying common web applications.
- AWS CloudFormation gives you the most control when you're deploying more complex applications.